C36

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44

A1

(11) International Publication Number: WO 96/18947

(43) International Publication Date: 20 June 1996 (20.06.96)

(21) International Application Number: PCT/US95/15959

(22) International Filing Date: 12 December 1995 (12.12.95)

(30) Priority Data: 08/355,356 13 December 1994 (13.12.94) US

(71) Applicant *(for all designated States except US)*: NOVELL, INC. [US/US]; 1555 North Technology Way, Orem, UT 84057 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: OLDS, Dale, R. [US/US]; 1623 East Edgecliff Drive, Sandy, UT 84092 (US).

(74) Agent: MESSENGER, Emamarie; Novell, Inc., 1555 North Technology Way, M/S A232, Orem, UT 84057 (US).

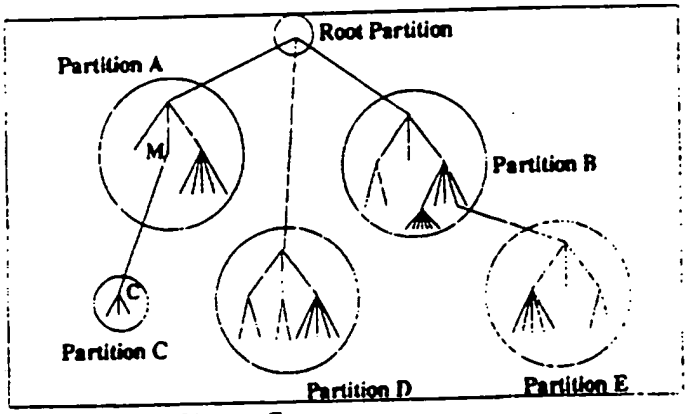
(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, LS, MW, SD, SZ, UG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS TO UPDATE OR CHANGE A NETWORK DIRECTORY



Typical Directory Tree

(57) Abstract

The present invention provides a method and apparatus for dynamically updating computer programs that are providing X.500 directory services without interruption of service. Upon receiving a call to update a computer program providing directory services, a process or thread is executed that authenticates the user making the reload request, loads a program loader, renames the then currently running directory services program, and calls another process or thread, while it awaits completion of the second process or thread. The second process or thread loads and initializes the new directory services computer program and then interacts with the program loader and the new directory services computer program to determine if the old and new directory services programs are compatible. If the old and new directory services computer programs are not compatible, the second process or thread aborts the load and transmits an abort signal to the first process or thread. If the old and the new directory services computer programs are compatible, the second process or thread transmit a commit signal to the first process or thread. The first thread upon receiving a commit signal from the second process or signal removes itself and the old directory services computer program from memory. Upon receiving an abort signal from the second process or thread, the first process or thread changes the name of the old directory services computer program back to its original name and unload itself and the loader program.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgyzstan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

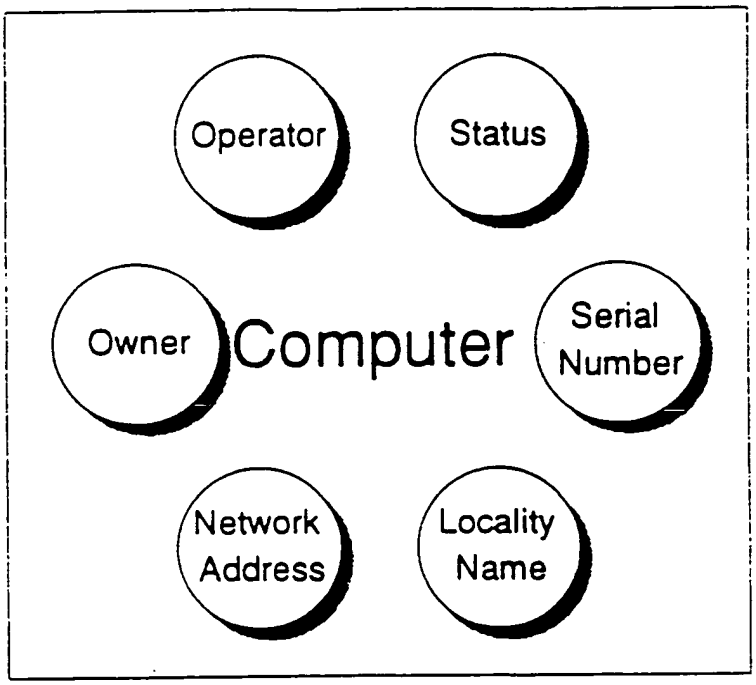


Figure 1 - Computer Object & Associated Attributes

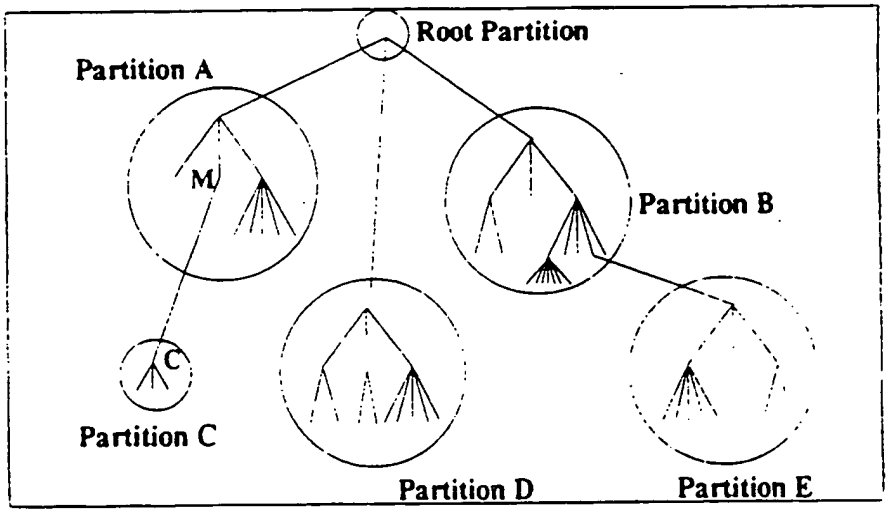


Figure 2 - Typical Directory Tree

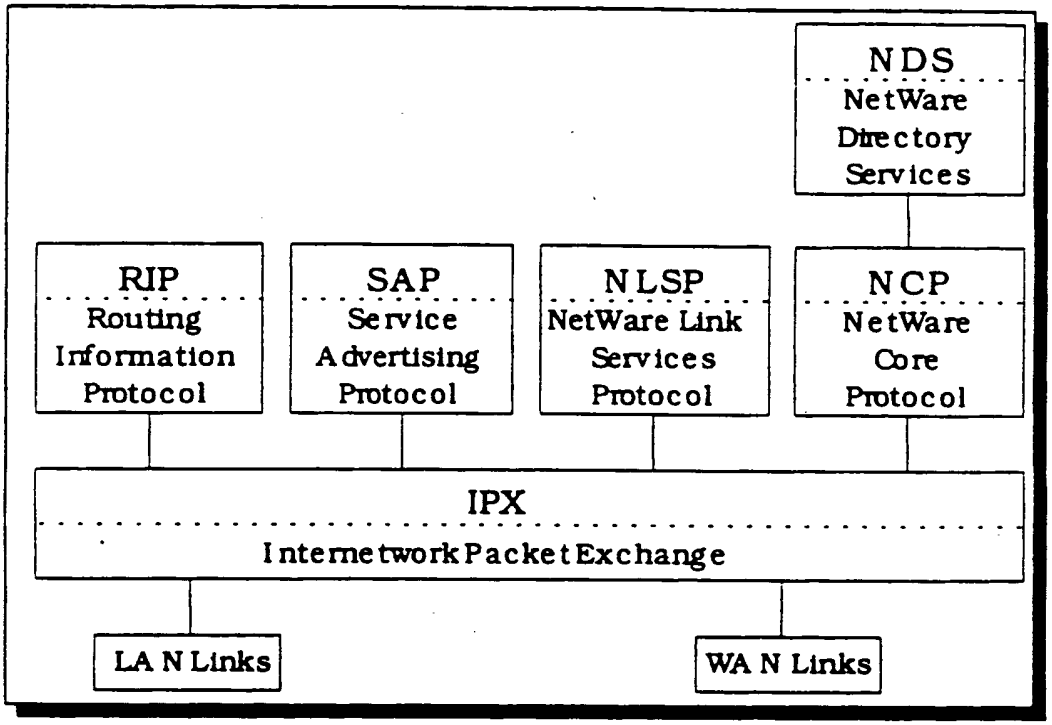


Figure 3 - NetWare Protocols

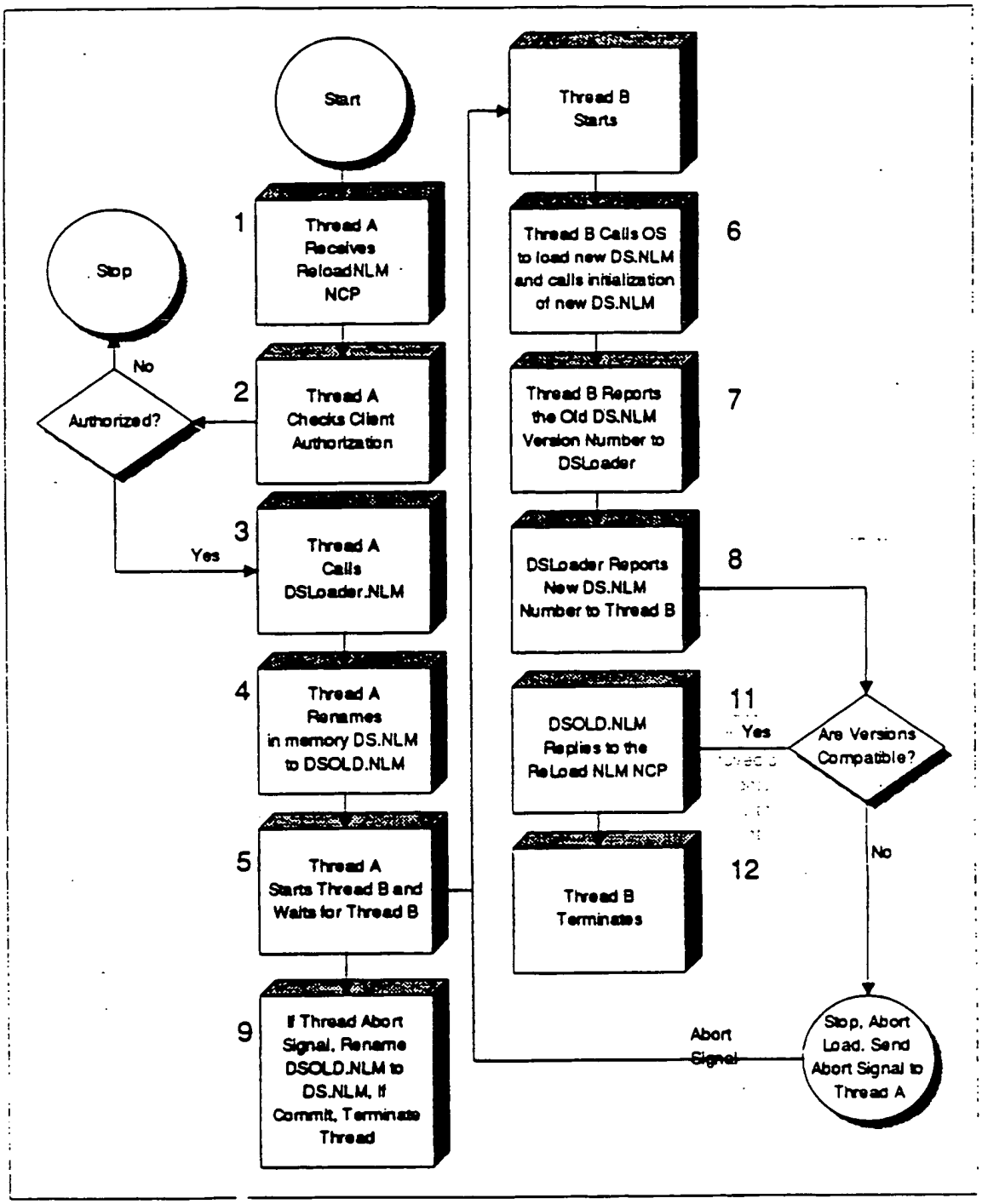


Figure 4 - Dynamic Update Algorithm

INTERNATIONAL SEARCH REPORT

International application No
PCT/US 95/15959

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON FAULT TOLERANT COMPUT (FTCS), TOULOUSE, JUNE 22 - 24, 1993, no. SYMP. 23, 22 June 1993, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, pages 30-35, XP000437223 DEEPAK GUPTA ET AL: "INCREASING SYSTEM AVAILABILITY THROUGH ON-LINE SOFTWARE VERSION CHANGE" see abstract see page 30, left-hand column, line 1 - line 19 see page 31, left-hand column, line 2 - line 10 see page 33, left-hand column, line 1 - right-hand column, line 33 --- -/--	1-16

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
8 May 1996	22.05.96

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax (+ 31-70) 340-3016	Michel, T

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 95/15959

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	DATA COMMUNICATIONS INTERNATIONAL, OCT. 1992, USA, vol. 21, no. 13, ISSN 0363-6399, pages 53-56, 58, 60, XP000569105 RUIZ B ET AL: "Netware 4.0: a directory to the enterprise" see page 56, middle column, line 3 - page 58, left-hand column, line 34 see page 58, right-hand column, line 27 - page 60, middle column, line 19 ---	1-16
A	US,A,5 155 847 (KIROUAC DONALD L ET AL) 13 October 1992 see abstract see column 1, line 54 - column 3, line 4; figure 1 ---	1-16
A	US,A,5 359 730 (MARRON ASSAF) 25 October 1994 see abstract see column 2, line 53 - column 4, line 2 ---	1-16
A	WO,A,94 01819 (ERICSSON TELEFON AB L M) 20 January 1994 see the whole document ---	1-16
A	EP,A,0 426 911 (HEWLETT PACKARD CO) 15 May 1991 see the whole document -----	1-16

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International licaeon No
PCT/US 95/15959

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
US-A-5155847	13-10-92	CA-A-	1310131	10-11-92
US-A-5359730	25-10-94	NONE		
WO-A-9401819	20-01-94	US-A-	5410703	25-04-95
		AU-B-	667559	28-03-96
		AU-B-	4516493	31-01-94
		EP-A-	0648353	19-04-95
		FI-A-	946195	30-12-94
		NO-A-	945096	22-02-95
EP-A-0426911	15-05-91	NONE		

Method & Apparatus To Update or Change A Network Directory

Background

The present invention relates to the management of distributed digital network directories, and particularly to providing dynamic updates to the computer programs supporting distributed directory services.

Technological advances in microelectronics and digital computing systems have resulted in the proliferation of digital computer networks, enabling the distribution of networking services across a wide range of computers participating in the network and over various communications media. Advances in distributing applications have also resulted in a client-server architecture for applications. Under the architecture, the portions of the application that interact with the user are typically separated from the portions of the application that fulfill client processing requests. Typically, the portions of an application that interact with the user are called a client applications or client software, whereas the portions of the application that service requests made by the client applications are called a server applications or server software. In a network environment, the client applications and server applications are generally executed on different computers.

Historically, digital networks in the form of local area networks, a physical collection of personal computers interconnected with network cabling and network interface cards, comprised a single network server and multiple network clients. To manage which network clients could access the network server, as well as what files, printers, printer queues, and server applications were available to the network clients, the network server maintained information on each of the resources that were attached to the server, the identities of the network clients and users who could use the services of the network server, and the scope and nature of the services available to the network clients and users.

As local area networks became more popular, networks grew in size requiring several servers to service the needs of users. With increased size and complexity of networks, came the need for easier management of network servers. Users required access to an increasing number of services that were located on an increasing number of network servers. Several vendors began

offering networking servers. Each vendor implemented a different scheme of providing networking services information. In addition, because of the way the server maintained information about only its networking services, each network server still required management of its resources independent of other network servers.

This insular method of maintaining information of networking services fueled research and development of distributed networking directories that span networking servers. Thus far, research has resulted in several potential solutions. Three technologies currently hold greater promise for replacing the large number of insular, idiosyncratic directories that now litter many an enterprise's numerous local-area networks and electronic-mail systems. One of the more popular approaches exploits the X.500 distributed network information directory services protocol developed as published by the CCIT and Open Systems Interconnect consortium.

However, while the X.500 protocol appears to hold the greatest promise to provide a robust, distributed directory, the X.500 protocol has been slow to gain acceptance. The X.500 protocol has been plagued from the start with management, interoperability and security problems. The X.500 protocol specification describes a technical framework, interoperability requirements and compliance criteria but does not describe specific implementations. Thus many of the details of implementation have been left up to systems providers.

The X.500 protocol specification describes a distributed directory. The directory provides information services to network clients. The information in the directory can be read as well as modified by users who have applicable access rights.

The information stored in the directory is stored in the form of a schema, a collection of objects with associated attributes or properties tied together by their relationship to each other. Figure 1 shows an object called "Computer" with a few associated attributes, such as owner, operator, status, etc. The values of the properties are not shown in the figure but an example of a value for "Owner" might be "Fred." Objects in the directory and their names correspond to things that humans relate to when dealing with computers, namely, users, printers, print queues, networks and information. Objects such as countries, organizations, networks, people and computers are objects you might find in the directory as well.

The directory provides information to users by giving users a hierarchical view of all of the information contained in the directory. The hierarchical view is generally in the form of a tree. Figure 2 shows a directory. Each of the branches and terminating points or leaves represent objects in the directory. Generally, implementations of the directory organize objects in subtrees, partitions or domains. Figure 2 also shows the directory organized into partitions or domains. Multiple copies of each partition may be stored in the directory. Software schemas define and determine the number and types of replicas of each partition.

Multiple replicas of a partition are needed to reduce network storage and traffic requirements and speed up directory searches. Replicas are stored in name servers. A name server is a computer in the network, usually a network server. More than one partition can be stored in a name server. Partitions stored in a name server need not be contiguous.

The directory tree provides a logical means of searching for information. The tree is generally patterned after logical groupings such as organizations, organizational units, computers and users. These logical groupings, while extremely useful in helping users find relevant information also creates significant problems in managing the directory.

Each partition forms a major subtree of the directory. Taken together, the partitions form a hierarchical tree of partitions that leads back to a root partition containing the root directory. Where boundaries of two partitions meet, the partition closer to the root is considered superior, and the partition farther from the root is considered subordinate. Thus, Figure 2, partitions E and C are subordinate to the other partitions.

The present invention solves one of the problems associated with a distributed directory. As distributed directories become more popular, more and more users will rely on them for access to data and services. As user rely on directories more heavily, the time in service of the directory will be critical. Users will not tolerate even a temporary shut down of the directory or a portion of the directory.

Summary of the Invention

With the present invention the computer programs that provide the services associated with a distributed directory can be dynamically updated without a significant interruption in services. Time in service of the directory will thus increase, increasing user confidence in the directory.

Brief Description of the Drawings

The present invention may be more fully understood by reference to the following Detailed Description in conjunction with the Drawings, in which:

Figure 1 shows a typical directory object, a computer, with some of its associated attributes;

Figure 2 shows a typical directory tree;

Figure 3 shows the network protocol environment in which the present embodiment of the invention is implemented; and

Figure 4 shows the software algorithm employed by the invention to dynamically update a directory services module without interruption of services.

Detailed Description of the Invention

The present embodiment of the invention, Novell's NetWare Directory Service or NDS, supports dynamically updating the computer programs that provide distributed digital directories. NDS operates in the NetWare network operating system environment.

The invention is enabled through a NetWare Core Protocol verb. NDS design builds on several previously implemented capabilities of NetWare, including the NetWare Core Protocol ("NCP"). The first capability relevant to the invention is NetWare's native network layer protocol, IPX. IPX provides end-to-end datagram delivery over network media and over internetworks.

NDS allows multiple independent name trees to coexist in the same internetwork without interfering with each other. A rendezvous feature is defined allowing a client interested in a name

tree to locate NDS name servers. The rendezvous feature builds on another previously implemented capability of NetWare: SAP (Service Advertising Protocol). Routers in all installed NetWare internetworks convey SAP information for client/server rendezvous. With NDS, SAP has a narrowly confined role: a client uses it to find its first NDS name server.

The NCP sits above the network layer. See Figure 3. NCP supports many networking services, such as file services. Certain operations on an NCP connection are specific to NDS. Once an NCP connection exists, it can also convey NDS requests and replies. Because NDS uses messages that can be quite large, it employs a fragmentation protocol to convey an NDS message in (possibly) several NCP packets.

Each NCP packet begins with a small message header that carries general status information about the current state of the connection between the client and the server. The client request header is seven bytes long, while a server's reply header is eight bytes long. As shown below, the RequestType variable defines the type of network request. A type of 0x1111 is reserved for connection allocation services; a type of 0x2222 is reserved for server request services; a type of 0x3333 is reserved for server responses; a type of 0x5555 is reserved for destroying connections; and a type of 0x9999 is reserved for work in progress responses.

Reload Verb
0x2222 104 8

Request Format

Offset	Content		Type
0	RequestType	(0x2222)	WORD
2	SequenceNumber	(LastSeq+1)	BYTE
3	ConnectionHigh	(ServiceConn)	BYTE
4	TaskNumber	(CurrentTaskNum)	BYTE
5	ConnectionLow	(ServiceConn)	BYTE
6	FunctionCode	(104)	BYTE
7	SubFuncCode	(08)	BYTE

Reply Format

Offset	Content		Type
0	Reply Type	(0x3333)	WORD
2	SequenceNumber	(LastSeq+1)	BYTE
3	ConnectionLow	(ServiceConn)	BYTE
4	TaskNumber	(CurrentTaskNum)	BYTE
5	ConnectionHigh	(ServiceConn)	BYTE
6	CompletionCode	(Ccode)	BYTE
7	ConnectionStatus	(StatusFlag)	BYTE
8	NDSErrorCode	(NDSError)	4 BYTES
12	Reserved		4 BYTES

The sequence number maintains a numeric counter for all incoming requests to provide reply prioritization. The ConnectionLow and the ConnectionHigh numbers identify a particular service connection between the client and the server. The TaskNumber distinguishes which client process or thread is making the request to the server.

The present embodiment of the invention uses the Reload Directory Services NCP. The Reload Directory Services NCP allows the principal computer program that provides directory services in the NetWare environment, DS.NLM, to be replaced on disk and reloaded in a server while that server is active and while other computer programs, NetWare Loadable Modules or NLMs in the NetWare environment, of the server are actively referencing NDS entry points.

Three NLMs are involved. The DSLOADER.NLM contains the directory entry points to which all other NLMs actually link, including the current DS.NLM in memory and a new DS.NLM on disk which is to replace the current DS.NLM.

Referring to Figure 4 and the code segments provided in Tables . . .. the dynamic update aspect of the invention is performed by two threads of execution within the NetWare operating system. The first thread (A) is the thread that begins servicing the NCP request, the other thread (B) is started by thread (A) to complete the reload of the new DS.NLM. The replacement algorithm is as follows:

1. Thread (A) receives the RELOAD NLM NCP request in a function that is part of the currently loaded DS.NLM.

2. Thread (A) checks the client authorization.

3. If the client has proper authorization, usually the highest level of security clearance allowed by the system, thread (A) calls the DSLOADER and requests a reload.

4. Thread (A) renames the memory image of the currently loaded DS.NLM to DSOLD.NLM.

5. Thread (A) starts thread (B) and then waits until thread (B) reports whether or not the load was successful.

6. Thread (B) calls the operating system to load the new DS.NLM. This loads the new DS.NLM and then calls DS.NLM's initialization function.

7. While initializing the new DS.NLM, thread (B) reports the new DS.NLM version number to DSLOADER and retrieves from DSLOADER the DSOLD.NLM version number. The DSLOADER may reject the load with an error response or it may return the new DS.NLM version number.

8. Thread (B) will abort the load on an error from the loader, or if the new DS.NLM rejects the version number returned by DSLOADER. Thread (B) will indicate to thread (A) if it aborts or commit to continue the load.

9. Thread (A) detects the abort or commit state transition from thread B. If the load is aborted thread (A) renames the DSOLD.NLM back to DS.NLM in memory. It then returns from the loader.

10. The DSOLD.NLM replies to the NCP request.

11. If Thread (B) commits to continue the load it waits for thread (A) to complete the response to the NCP, then it will unload DSOLD.NLM and continue with the initialization of the new DS.NLM.

12. Thread B terminates itself.

Table 1 - Code Segments Implementing Steps 1-3 and 10

```
int ReloadDS(int conn)
(
      int err = 0, managesEntry;
      THREADDATA td;

      /* 1. begin servicing reload NCP */
      if (err = DSAClientStart(TD_CHECK_OPEN, conn, -1, -1, &td))
            return err;

      /* 2. check client authorization */
      if (IsSupervisor(conn)
                  || !(err = GlobalCheckManagement(ServerID(), ID_SELF,
&managesEntry, 0))
                  && managesEntry)
            err = DSLReload(DSModuleHandle());  /* 3. call loader */
      else if (!err)
            err = ERR_NO_ACCESS;

      /* 10. reply to the NCP */
      return DSAClientEnd(err);
```

Table 2 - Code Segments Implementing Steps 4-5 and 9

```
int DSLReload(uint32 moduleHandle)
{
      /* 4. rename DS.NLM to DSOLD.NLM */
      struct LoadDefinitionStructure *mh = (struct LoadDefinitionStructure
*)moduleHandle;
      char savedName[sizeof(mh->LDFileName)];
      if (!mh)
            mh = (struct LoadDefinitionStructure *)registeredModule;
      else if (moduleHandle != registeredModule)
            return ERR_INVALID_REQUEST;

      if (dslState != DSL_IDLE)
            return ERR_DS_LOADER_BUSY;
      dslState = DSL_ACTIVE;
      if (mh)
      {
            /* actual rename happens here */
            CMovB(mh->LDFileName, savedName, sizeof(savedName));
            CMovB(dyingNLMName, mh->LDFileName, sizeof(savedName));
      }

      /* 5. start new thread and wait for state change */
      aesReload.AWakeUpDelayAmount = 0;
      aesReload.AProcessToCall = ReloadWorker;
      aesReload.ARTag = aesTag;
      ScheduleSleepAESProcessEvent(&aesReload);
      while (dslState == DSL_ACTIVE)
            CYieldWithDelay();


      /* 9. detect state and rename DSOLD.NLM back to DS.NLM if abort */
      if (dslState == DSL_ABORTED)
      {
            if (mh)
            {
                  CMovB(savedName, mh->LDFileName, sizeof(savedName));
                  /* mh->LDFlags |= oldDontUnloadBit; */
            }
            dslState = DSL_IDLE;
            return loadError;
      }
      if (dslState == DSL_PROCEEDING)
            dslState = DSL_COMMITTED;
      return 0;
}
```

Table 3 - Code Fragments Implementing Step 6 and part of 8

```
void ReloadWorker(void)
{
      unsigned long oldModule = registeredModule;
      if (dslState != DSL_ACTIVE)
            return;
      /* 6. call load function */
      if (loadError = LoadDSNLM())
      {
            /* 8. indicate the abort state change */
            if (dslState == DSL_ACTIVE)
                  dslState = DSL_ABORTED;
      }
      else if (dslState == DSL_CORPSE)
      {
            if (oldModule)
            {
                  DelayMyself(18, timerTag);
                  KillMe(oldModule);
            }
            dslState = DSL_IDLE;
      }
}
```

Table 4 - Code Fragments Implementing Step 7 and the Remainder of Step 8

```
int RegisterWithDSLoader(void)
{
      int err;
      uint32 dslVersion, loadedDSVersion;
      int i;

      /* ... */

      /* 7. negotiate versions with loader. Handle loader's rejection */
      if (err = DSLNegotiateVersions(DSVersion(), &dslVersion,
&loadedDSVersion))
            return err;

      if (!ACCEPTABLE_DSLOADER_VERSION(dslVersion))
            return ERR_INVALID_DS_VERSION; /* reject the loader */

      /* 8. commit to load new NLM, this changes loader state */
      if (err = DSLRegister(DSModuleHandle(), DSVersion(), &ddsFuncs,
&emuFuncs,
                  DSCanUnload, DSUnload, &dslMemTag, dslCommandLine))
            return err;

      /* continue NLM initializations */
      /* ... */

      return 0;
}
```

Table 5 - Code Fragments Implementing DSLoader Response to Step 7

```
int DSLNegotiateVersions(uint32 dsVersion, uint32 *dslVersion,
            uint32 *registeredDSVersion)
{
      *dslVersion = INTERNAL_VERSION;
      *registeredDSVersion = registeredVersion;
      dsVersion = dsVersion;
      return ACCEPTABLE_DS_VERSION(dsVersion)? 0: ERR_INVALID_DS_VERSION;
}
```

As indicated by the above method, the computer programs providing services to a distributed directory can be dynamically updated without interruption of directory services. Thus, critical directory related services can be updated and new service enhancements can be added without interruption.

Although one embodiment of the invention has been illustrated and described, various modifications and changes may be made by those skilled in the art without departing from the spirit and scope of the invention.

Claims

1. A method, in a computer network, of dynamically updating an old computer module with a new computer, comprising the steps of:
   a. receiving a request to update an old computer module;
   b. calling a loader computer module, which routes requests from the old computer module, having the basic functionality of the old computer module;
   c. loading a new computer module to replace the old computer module; and
   d. making active the new computer module and making inactive the old computer module.

2. A method as recited in claim 1, whereby the new computer module is a more current version of the old computer module.

3. A method as recited in claim 1, further comprising the step of: checking that the request to update has valid authorization prior to calling the loader computer module.

4. A method as recited in claim 1, further comprising the step of: checking that the new computer module is compatible with the old computer module prior to unloading the old computer module.

5. A method as recited in claim 1, whereby the old and new computer modules are NetWare loadable modules.

6. A method as recited in claim 1, whereby the old and new computer modules provide directory services.

7. A method, in a computer network, of dynamically updating an old computer module with a new computer module, comprising the steps of:
   a. receiving a request to update an old computer module;
   b. calling a loader computer module, which routes requests from the old computer module, having the basic functionality of the old computer module;
   c. loading a new computer module to replace the old computer module;

d. checking that the new computer module is compatible with the old computer module; and

e. making active the new computer module and making inactive the old computer module if the new computer module is compatible with the old computer module.

8. A method as recited in claim 7, whereby the new computer module is a more current version of the old computer module.

9. A method as recited in claim 7, further comprising the step of: checking that the request to update has valid authorization prior to calling the loader computer module.

10. A method as recited in claim 7, whereby the old and new computer modules are NetWare loadable modules.

11. A method as recited in claim 7, whereby the old and new computer modules provide directory services.

12. A method of dynamically updating an old computer module being used on a server in a client/server network with a new computer module, comprising the steps of:

a. receiving a request to update an old computer module;

b. calling a loader computer module, which routes requests from the old computer module, having the basic functionality of the old computer module;

c. loading a new computer module to replace the old computer module;

d. checking that the new computer module is compatible with the old computer module; and

e. making active the new computer module and making inactive the old computer module if the new computer module is compatible with the old computer module.

13. A method as recited in claim 12, whereby the new computer module is a more current version of the old computer module.

14. A method as recited in claim 12, further comprising the step of: checking that the request to update has valid authorization prior to calling the loader computer module.

15. A method as recited in claim 12, whereby the old and new computer modules are NetWare loadable modules.

16. A method as recited in claim 12, whereby the old and new computer modules provide directory services.